Mutations

A *single mutation* of a sequence [a1,a2,…am] is a cyclic permutation of a subset of those elements, subject to certain constraints; a *compound mutation* is a set of disjoint single mutations (i.e. single mutations of disjoint subsets of elements). For notational convenience a single mutation is represented by a list in parentheses, beginning with its smallest element and ending with a repeat of the same element. The constituent single mutations in a compund mutation should then be written in descending order of their first element. A self-cycle or *singleton* is written as a pair thus: .

An element in a mutation may be *barred*, that is, written with a bar over it. For this to be permitted it must be followed by a smaller number. For computer purposes a barred number can be represented by its negative. A mutation without any barring (except where that isn’t permitted – see rule 3 in the next section) is called *plain.*

The following rules derive from the meaning of the mutation. In the graceful tree problem a mutation represents a number of changes to a base graph, which is a star graph with the 0-vertex at its hub. So (1 2 1) will detach the 1- and 2-vertices from the hub and attach them to the vertex labelled with their sum, in this case the 3-vertex:

This is achieved by starting with the zero Cantor list, [0,0,0,0], and then reading the mutation as “1 in the 2-place, 2 in the 1-place”, which gives [2,1,0,0]. The 1-place, here, means the left-most position in the Cantor list, the 2-place is the next one along, and so on. In the case of a barred entry – say if it read – the previous entry still goes into the same place as before, but the barred entry puts the *difference* between the barred number and its successor into the successor’s place. This gives us “1 in the 2-pace, 2-bar in the 1-place”, or [1,1,0,0].

# Mutation Rules

The constraints referred to in paragraph one are as follows:

1. If is the number of edges in the tree, only the numbers between 0 and (exclusive) [[1]](#footnote-1)can be used in a mutation;
2. In singleton mutations the element must not exceed
3. Iff an element in a mutation is greater than the following element, it may optionally be barred. If the two sum to more than barring is compulsory;
4. Except when the first of the two is barred, each pair of consecutive elements in a single mutation must sum to no more than

The integer equivalent of a Cantor list is called the tree’s *index*. Disjoint mutations generate Cantor lists that can be combined by adding corresponding list elements together, or by just adding their indices. The trees are called “orthogonal” to one another, and if one has a particular vertex not adjacent to the 0-vertex, the corresponding vertex in the other tree *must* be 0-adjacent.

When we have generated all the valid single mutations on edges, we can then make compound mutations by packaging together sets of pairwise disjoint mutations.

# Generating Mutations

The method is as follows:

1. Generate all the singleton mutations - , , and so on, up to . Save them as “size 1 mutations”, labelling each one with its Cantor index.
2. For each existing plain mutation in the “size n” list, pick a number larger than the largest element of that mutation. Try it in each place (only including barred entries when barring them is compulsory). If the number is permitted there, save the resulting plain mutation in “size n+1 mutations” (labelled with its Cantor index), and continue testing the remaining places, until all numbers have been tested in all places in all the size n mutations.
3. Repeat for the set of size n+1 mutations to generate a set of size n+2 mutations, and so on.
4. Once the resulting set of size N+1 mutations is empty, stop.
5. You now have N sets of plain mutations, from size 1 to size N.
6. Go through every mutation finding any elements amenable to barring. If a mutation has k such loci it will have versions with every combination of barred and unbarred (including the original plain version). Calculate each one and save it (with its index) with the plain original.
7. Now go through all the mutations finding which ones are disjoint from each other. Bundle them into compound mutations. Save them, labelled with the sum of their indices.

Here are some projected results. I haven’t formally proved any of them yet, but – where I can – I’ve sketched how I would *approach* proving them.

1. **This will produce every valid mutation**I would prove this by descent – any mutation can have its largest element deleted, leaving a smaller valid mutation. This can be repeated until only a singleton remains. So any valid mutation can be tracked back to a singleton (the smallest number, which bookends the list), and then the above procedure will run that process in reverse.
2. **Every tree produced by this method is graceful**   
   By induction, if I can show that every introduction of a new element in the mutation (complying with the rules) will preserve the property of gracefulness, I’m pretty much there, since the initial star graph is known to be graceful.
3. **Every graceful tree can be produced by this method**I plan to attempt a constructive proof – I have an algorithm to take a Cantor index and return the corresponding mutation, which should fail if the graph is cyclic.
4. **Every tree (up to isomorphism) will be represented**. I have no idea how to prove this yet. It entails the graceful tree conjecture, so I expect it to be super-hard. An early part of the process will be about enumerating mutations. This is already hard!

1. If you include your set of mutations will include those of odd Cantor index. As these are only the converse labellings of the even-index ones, I have decided to exclude them to save unneccessary work. [↑](#footnote-ref-1)